Squaring the Triangle: Secure, Decentralized, Human-Readable Names

When using computers, we like to refer to things with names. For example, this website is known as “www.aaronsw.com”. You can type that into your browser and read these words. There are three big properties we might want from such names:

In a classic paper, my friend Zooko argued that you can get at most two of these properties at any one time.

Recently, DNS legend Dan Kaminsky used this to argue that since electronic cash was pretty much the same as naming, Zooko’s triangle applied to it as well. He used this to argue that BitCoin, the secure, decentralized, human-meaningful electronic cash system was impossible. I have my problems with BitCoin, but it’s manifestly not impossible, so I just assumed Kaminsky had gone wrong somewhere.

But tonight I realized that you can indeed use BitCoin to square Zooko’s triangle. Here’s how it works:

Let there be a document called the scroll. The scroll consists of a series of lines and each line consists of a tuple (name, key, nonce) such that the first N bits of the hash of the scroll from the beginning to the end of a line are all zero. As a result, to add a line to the scroll, you need to do enough computation to discover an appropriate nonce that causes the bits of the hash to be zero.

To look up a name, you ask everyone you know for the scroll, trust whichever scroll is the longest, and then start from the beginning and take the key for the first line with the name you’re looking up. To publish a name, you find an appropriate nonce and then send the new line to everyone you know.

OK, let’s pause there for a second. How do you steal names in such a system? First, you need to need to calculate a new nonce for the line you want to steal and every subsequent line. Second, you need to get your replacement scroll to the user. The first is difficult, but perhaps not impossible, depending on how many lines ago the name you want to steal is. It requires having some large multiple of the rest of the network’s combined CPU power. This seems like a fairly strong constraint to me, but apparently not to Dan. Luckily, we’re saved by the second question.

Let there be a group of machines called the network. Each remembers the last scroll it trusted. When a new valid line is created it’s sent to everyone in the network and they add it to their scroll.[1] Now stealing an old name is impossible, since machines in the network only add new names, they don’t accept replacements for old ones.

That’s fine for machines already in the network, but how do you join? Well, as a physical law, to join a network you need the identity of at least one machine already in the network. Now when you join, that machine can give you a fabricated scroll where they’ve stolen all the names. I don’t think there’s any way to avoid this — if you don’t know anyone willing to tell you the correct answer, you can’t will the correct answer out of thin air. Even a centralized system depends on knowing at least one honest root.

You can ameliorate this problem by knowing several nodes when you connect and asking each of them for their scroll. It seems like the best theoretically-possible case would be requiring only one node to be honest. That would correspond to trusting whichever node had the longest scroll. But this would leave you vulnerable to an attacker who a) has enough CPU power to fabricate the longest scroll, and b) can co-opt at least one of your initial nodes. The alternative is to trust only scrolls you receive from a majority of your list of nodes. This leaves you vulnerable to an attacker who can co-opt a majority of your initial nodes. Which tradeoff you pick presumably depends on how much you trust your initial nodes.

Publishing a false scroll is equivalent to fragmenting the namespace and starting a separate network. (We can enforce this by requiring nodes to sign each latest scroll and publish their signature to be considered members-in-good-standing of the network. Any node that attempts to sign two contradictory scroll is obviously duplicitous and can be discounted.) So another way of describing scenario (b) is to say that to join a network, you need a list of nodes where at least a majority are actually nodes in the network. This doesn’t seem like an overly strenuous requirement.

And we’re actually slightly safer than that, since the majority needs a fair amount of CPU to stay plausible. If we assume that you hear new names from some out-of-band source, for them to work on the attacker’s network, the attacker must have enough CPU to generate lines for each name you might use. Otherwise you realize that the names you type in on your computer are returning 404s while they work on other people’s computers and begin to realize you’ve been had by an attacker.

So there you have it. The names are secure: they’re identifiable by a key of arbitrary length and cannot be stolen. They’re human-meaningful: the name can be whatever string you like. And they’re decentralized: no centralized authority determines who gets what name and yet they’re available to everyone in the network.

Zooko’s triangle has been squared.

UPDATE: I’m gratified by all the feedback and I’ve put up a Frequently Asked Questions page in response to comments here and elsewhere.

What happens if two people create a new line at the same time? The debate should be resolved by the creation of the next new line — whichever line is previous in its scroll is the one to trust. ■

You should follow me on twitter here.

January 6, 2011